***Software Engineering Assignments***

**1. What is a Program?**

* A program is a collection of instructions written in a programming language that a computer can execute to carry out a specific task or function. Essentially, a program tells the computer what to do, step-by-step, using a specific syntax and logic. It can take input, process it, and produce output.
* Programs are written using programming languages (e.g., C, Python, Java), which provide a way for humans to communicate with computers. These instructions are usually executed in sequence, but they can also involve loops, conditionals, and more complex structures depending on the task.

**LAB EXERCISE: Simple "Hello World" Program in C**

#include <stdio.h>

#include<conio.h>

int main()

{

printf("Hello, World!\n");

return 0;

}

**2. What is Programming?**

Programming is the process of designing, writing, testing, and maintaining a set of instructions (called a program) that a computer can understand and execute to perform a specific task. It involves using a programming language (like C, Python, Java, etc.) to solve problems or automate tasks.

In simple words, programming is how we communicate with computers to tell them what to do.

**THEORY EXERCISE**: What Are the Key Steps Involved in the Programming Process?

The programming process involves several important steps. Here are the key steps in order:

**1. Understanding the Problem**

Before writing any code, it's important to clearly understand what the problem is.

Ask: What do I want the program to do? What input is required? What should be the output?

**2. Planning the Solution (Algorithm Design)**

Think about how to solve the problem step-by-step.

This is often done by creating an algorithm (a sequence of steps) or writing a flowchart or pseudocode.

**3. Writing the Code (Implementation**)

Translate the algorithm into actual code using a programming language (e.g., C, Python, Java).

This is where you start coding the solution.

**4. Compiling (for compiled languages like C)**

If you're using a compiled language like C or C++, you must compile your code using a compiler.

The compiler converts your human-readable code into machine-readable code.

**5. Testing and Debugging**

Run the program and check if it gives the correct output.

If there are errors (called bugs), you need to debug the code and fix them.

**6. Documentation**

Write comments and documentation so others (or future you) can understand your code.

Good documentation helps in maintaining and updating the program later.

**7. Maintenance and Updates**

After the program is running, it may need updates or improvements over time.

This includes fixing bugs found later or adding new features.

**3. Types of Programming Languages**

There are two types of programming language:

**1. High-level language**

High-level languages are designed to be easy for humans to understand and use, making them ideal for application development.

**2.Lower-level language**

Low-level languages are closer to machine code, offering greater control and efficiency, which is important for system-level programming.

**THEORY EXERCISE: What are the main differences between high-level and low-level programming languages?**

Understanding the differences between high-level and low-level programming languages is fundamental in computer science. Here's a clear comparison:

**Feature : Abstraction**

High-Level Programming Languages : High abstraction from hardware

Low-Level Programming Languages: Little or no abstraction from hardware

**Feature : Ease of Use**

High-Level Programming Languages : Easier to read, write, and understand

Low-Level Programming Languages : Harder to read and write

**Feature : Portability**

High-Level Programming Languages : Portable across platforms

Low-Level Programming Languages : Not portable (tied to specific hardware/CPU)

**Feature : Speed/Performance**

High-Level Programming Languages : Slower execution (but faster to develop)

Low-Level Programming Languages : Faster execution

**Feature : Memory Management**

High-Level Programming Languages : Automatic (e.g., garbage collection)

Low-Level Programming Languages : Manual

**Feature : Control over Hardware**

High-Level Programming Languages : Limited direct access

Low-Level Programming Languages : Full control over hardware resources

**Feature : Compilation/Interpretation**

High-Level Programming Languages : Needs to be compiled or interpreted

Low-Level Programming Languages : Machine code runs directly

**Feature : Usage**

High-Level Programming Languages : Application software, web development, etc.

Low-Level Programming Languages : Operating systems, embedded systems, drivers

These are the main differences between high-level programming language and Low-level programming language.

**4. World Wide Web & How Internet Works**

**LAB EXERCISE: Research and create a diagram of how data is transmitted from a client to a server over the internet.**

Here's a step-by-step flow of data transmission over the internet, followed by a diagram:

Steps:

**Step 1:**

**Client enters a URL** (e.g., <www.example.com>

) in a web browser.

**Step 2:**

**DNS Resolution** The domain name is sent to a DNS server to get the IP address of the web server.

**Step 3:**

**Client sends an HTTP request** to the server's IP address via the Internet (using TCP/IP protocol).

**Step 4:**

**Routers and switches** forward the request through multiple networks to reach the server.

**Step 5:**

**Server receives the request**, processes it, and sends back an HTTP response (e.g., a webpage).

**Step 6:**

The response travels **back to the client** along a similar route.

**Step 7:**

The **web browser renders the content** for the user to see.

* **Diagram (Text-Based):**

[Client Browser]

|

v

[DNS Request] ---> [DNS Server] ---> IP Address

|

v

[HTTP Request]

|

v

[Internet Routers/Switches]

|

v

[Web Server]

|

v

[HTTP Response]

|

v

[Internet Routers/Switches]

|

v

[Client Browser Displays Webpage]

**THEORY EXERCISE: Describe the roles of the client and server in web communication.**

**Client :**

- A device or application (like a web browser) that initiates communication by sending requests to a server.

- Asks for services such as web pages, files, or data.

**Server :**

- A powerful computer or program that waits for incoming requests from clients.

- Processes the request and sends back the appropriate response (like HTML files, images, or JSON data).

**Example:**

* When you open <www.google.com>:
* **Client =** Your web browser
* **Server =** Google’s web server that sends you the search page

**5. Network Layers on Client and Server**

**LAB EXERCISE: Design a simple HTTP client-server communication in any language.**

**create:**

1. A server that listens on a port and responds with a basic HTTP message.

2. A client that connects to the server and prints the response.

**1. HTTP Server in C :**

// server.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char http\_response[] =

"HTTP/1.1 200 OK\r\n"

"Content-Type: text/html\r\n\r\n"

"<html><body><h1>Hello from C Server!</h1></body></html>\r\n";

// Creating socket file descriptor

server\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

// Set up the address struct

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

// Bind the socket

bind(server\_fd, (struct sockaddr\*)&address, sizeof(address));

// Listen for connections

listen(server\_fd, 3);

printf("Server listening on [http://localhost:%d\n](http://localhost:%25d\n)", PORT);

// Accept incoming connection

new\_socket = accept(server\_fd, (struct sockaddr\*)&address, (socklen\_t\*)&addrlen);

// Send the HTTP response

send(new\_socket, http\_response, strlen(http\_response), 0);

printf("HTTP response sent to client.\n");

close(new\_socket);

close(server\_fd);

return 0;

}

**2. HTTP Client in C:**

// client.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

char buffer[4096] = {0};

char \*http\_request = "GET / HTTP/1.1\r\nHost: localhost\r\n\r\n";

sock = socket(AF\_INET, SOCK\_STREAM, 0);

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 addresses from text to binary form

inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr);

// Connect to the server

connect(sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr));

// Send HTTP GET request

send(sock, http\_request, strlen(http\_request), 0);

// Read server response

read(sock, buffer, sizeof(buffer));

printf("Server response:\n%s\n", buffer);

close(sock);

return 0;

}

**How to Compile and Run:**

# Compile both programs

gcc server.c -o server

gcc client.c -o client

# Run the server in one terminal

./server

# Run the client in another terminal

./client

**THEORY EXERCISE: Explain the function of the TCP/IP model and its layers:**

The 4 Layers of the TCP/IP Model:

**1. Application Layer**

**Function:**

Provides network services directly to the user or applications (like web browsers, email clients).

**What it does:**

Formats and sends data (like web pages or emails).

Uses protocols like HTTP, HTTPS, FTP, DNS, SMTP, POP3.

**Example:**

When you open a website in your browser, HTTP is used to request and receive web pages.

**2. Transport Layer**

**Function:**

Provides end-to-end communication, ensuring data is delivered reliably and in order.

**Key Protocols:**

**TCP (Transmission Control Protocol):** Reliable, ordered, connection-oriented.

**UDP (User Datagram Protocol):** Faster, but connectionless and not reliable.

**What it does:**

1. Breaks large messages into segments.

2. Ensures each segment is delivered and reassembled at the destination.

**Example:**

-->TCP is used for loading websites.

-->UDP is used for streaming videos or online games.

**3. Internet Layer**

**Function:**

Handles routing and addressing so that data can travel across different networks.

Key Protocols:

IP (Internet Protocol) — defines IP addresses and routes packets.

ICMP (used for error messages like "Destination Unreachable").

**What it does:**

1.Breaks segments into packets.

2.Assigns IP addresses to source and destination.

3.Routes packets between networks.

**4. Network Access Layer (also called Link Layer)**

**Function:**

Deals with the physical transmission of data over the network hardware (like cables, routers, Wi-Fi).

**What it does:**

-->Converts packets into frames for transmission.

-->Uses MAC addresses.

-->Interacts with physical media (e.g., Ethernet, Wi-Fi).

**Example:**

Sending data over a LAN or wireless connection.

**Data Flow Summary (From Sender to Receiver)**

Application Layer → Your app prepares data (e.g., HTTP request)

Transport Layer → Data is broken into segments (TCP/UDP)

Internet Layer → Segments become packets, routed by IP

Network Access Layer → Packets become frames, sent over the physical network

**6. Client and Servers**

THEORY EXERCISE: Explain Client Server Communication

**Definition:**

Client-server communication is a model in computer networking where a client (user device or application) sends a request to a server, and the server processes that request and sends back a response.

This is the core concept behind most online services such as websites, emails, file downloads, and more.

**How Client-Server Communication Works:**

**1. The Client:**

A client is a device (like a computer or phone) or software (like a browser or app) that:

--> initiates the communication.

--> Sends a request for data or services.

--> Waits for and receives the response.

**2. The Server:**

A server is a powerful computer or software that:

--> Listens for incoming requests from clients.

-->Processes the requests.

-->Sends back the appropriate response (data, web page, file, etc.).

**Communication Process (Step-by-Step):**

1. Client sends a request (e.g., HTTP request to open a web page).

2. Server receives the request, processes it (e.g., retrieves the HTML file).

3. Server sends a response back to the client.

4. Client receives and uses the response (e.g., displays the webpage).

**Example (Web Browser and Web Server):**

Client: Web browser like Chrome or Firefox

Server: Website’s hosting server (e.g., <www.example.com> )

**Process:**

-->Client sends: GET /index.html.

-->Server responds: Sends back the HTML page content.

**7. Types of Internet Connections**

**LAB EXERCISE:**

**Research different types of internet connections (e.g., broadband, fiber, satellite) and list their pros and cons.**

**After research , there are 6 types of internet connection found:**

1. DSL (Digital Subscriber Line)

2. Cable Internet

3. Fiber-Optic Internet

4. Satellite Internet

5.Mobile Internet (4G/5G)

6. Dial-Up (Old Technology)

**Internet Connections Pros and Cons :**

**1. DSL (Digital Subscriber Line)**

**Uses:** Phone lines (without interrupting calls)

**Pros:**

1. Widely available

2. Cheaper than fiber or cable

**Cons:**

1. Slower speeds

2. Distance from provider affects performance

3. Fiber-Optic Internet

**2. Cable Internet**

**Uses:** Coaxial TV cables

**Pros:**

1. Faster than DSL

2.Widely available in cities

**Cons:**

1. Shared bandwidth = slower during peak hours

2. More expensive than DSL

3. Fiber-Optic Internet

3. Fiber-Optic Internet

**3. Fiber-Optic Internet**

**Uses:** Light signals through glass fibers

**Pros:**

1. Extremely fast speeds (up to 1 Gbps or more).

2. Reliable and consistent.

**Cons:**

1. Limited availability (mostly urban areas)

2. Expensive installation in rural areas

**4. Satellite Internet**

**Uses:** Satellite signals (no cables needed)

**Pros:**

1. Available almost everywhere (rural/remote areas)

**Cons:**

1. High latency (delay)

2. Weather can affect performance

3. Limited data plans

**5.Mobile Internet (4G/5G)**

**Uses: Cellular networks**

**Pros:**

1. Portable and wireless

2. Good speeds, especially with 5G

**Cons:**

1. Data limits or higher cost

2. Signal strength depends on location

**6. Dial-Up (Old Technology)**

**Uses: Traditional phone lines**

**Pros:**

1. Very cheap

**Cons:**

1.Extremely slow

2.Not suitable for modern usage

**THEORY EXERCISE: How does broadband differ from fiber-optic internet?**

**Broadband:**

**Definition:**

Broadband is a general term for high-speed internet connections that are always on and faster than traditional dial-up.

**Types of Broadband include:**

**1. DSL**

**2. Cable**

**3. Satellite**

**4. Fiber-optic**

**5. Mobile (4G/5G)**

**6. Speed & Technology:**

Speeds vary depending on the type (e.g., DSL is slower, cable is moderate).

**Fiber-Optic Internet:**

**Definition:**

Fiber-optic internet is a specific type of broadband that uses glass or plastic cables to transmit data using light signals.

**Speed & Reliability:**

1. Much faster than DSL or cable (up to 1 Gbps or more)

2. More reliable and stable connection

3. Low latency (delay)

Aspect Broadband Fiber-Optic Internet

Meaning General term for high-speed internet Specific type of broadband

Technology Uses copper wires, coaxial cables, etc. Uses glass fibers and light signals

Speed Varies (DSL: slow, cable: medium) Very fast (up to 1 Gbps or more)

Reliability Moderate Very high

Availability Widely available Limited to areas with fiber networks

**8. Protocols**

**LAB EXERCISE: Simulate HTTP and FTP requests using command line tools (e.g., curl).**

**1. Simulate an HTTP Request with curl**

curl <http://example.com>

**Explanation:**

This command sends an HTTP GET request to the server at example.com.

It retrieves and displays the HTML content of the homepage.

**2. Simulate an HTTPS Request with curl**

curl <https://example.com>

**Explanation:**

This sends a secure HTTP (HTTPS) request to the same server.

The connection is encrypted using SSL/TLS, protecting data from interception.

**3. Simulate an FTP Request with curl**

curl <ftp://speedtest.tele2.net/1KB.zip> --user anonymous:

**Explanation:**

This connects to a public FTP server and downloads a test file.

The --user anonymous: option is used for anonymous FTP login.

**THEORY EXERCISE: What are the differences between HTTP and HTTPS protocols?**

**Full Form HyperText Transfer Protocol HyperText Transfer Protocol Secure.**

Security Not secure (data sent in plain text) Secure (data encrypted with SSL/TLS)

Encryption No encryption Uses encryption

Port Number 80 443

URL Prefix http:// https://

Data Protection Vulnerable to attacks (e.g., sniffing) Protects data from hackers and spies

Browser Indicator No lock icon in address bar Lock icon 🔒 shows secure

connection

Use Cases Basic websites, blogs Banking, e-commerce, login pages

**9.Application Security**

**LAB EXERCISE: Identify and explain three common application security vulnerabilities. Suggest possible solutions.**

**1. SQL Injection (SQLi)**

**Explanation:**

Happens when attackers insert malicious SQL code into input fields (e.g., login forms) to access or modify a database.

**Example:**

Inputting ' OR '1'='1 in a login field to bypass authentication.

**Solution:**

Use parameterized queries or prepared statements.

Sanitize and validate all user input.

Avoid directly inserting user input into SQL queries.

**2. Cross-Site Scripting (XSS)**

**Explanation:**

Occurs when attackers inject malicious scripts into web pages that are viewed by other users. These scripts can steal session cookies or manipulate the DOM.

**Example:**

Posting <script>alert('Hacked')</script> in a comment section.

**Solution:**

1. Escape or encode output before displaying user input in HTML.

2. Use Content Security Policy (CSP) headers.

3. Filter and validate input from all sources.

**3. Insecure Authentication**

**Explanation:**

Weak login mechanisms (like simple passwords or no-account lockout) can allow brute-force or credential stuffing attacks.

**Solution:**

1.Enforce strong password policies.

2. Implement multi-factor authentication (MFA).

3. Use rate limiting and account lockout after failed attempts.

**THEORY EXERCISE: What is the role of encryption in securing applications?**

**Definition of Encryption:**

Encryption is the process of converting plain text into unreadable code (cipher text) so that only authorized users can read it using a decryption key.

**Role of Encryption in Application Security:**

**1. Protects Sensitive Data:**

1. Encrypts user data (e.g., passwords, credit card numbers) in storage and during transmission.

2. Prevents unauthorized access even if data is stolen.

**2. Secures Communication:**

1. SSL/TLS encryption ensures safe data exchange between client and server (used in HTTPS).

2. Prevents attackers from eavesdropping or tampering with messages.

**3. Ensures Data Integrity and Confidentiality:**

1. Makes sure data is not altered in transit.

2. Keeps information private and secure from third parties.

**4. Compliance with Regulations:**

1. Many laws (like GDPR, HIPAA) require encryption for handling personal or health data.

**Application Part** **Encryption Used**

Web communication SSL/TLS (HTTPS)

Password storage Hashing (e.g., bcrypt, SHA-256)

Database fields AES encryption for sensitive data

Encryption is essential for protecting data in applications. It helps ensure privacy, secure communication, and compliance with security standards — making it a fundamental part of modern application security.

**10. Software Applications and Its Types**

**LAB EXERCISE: Identify and classify 5 applications you use daily as either system software or**

**Identify and Classify 5 Daily-Use Applications as System Software or Application Software**

**Software/Application Type Why?**

Google Chrome Application Software Used for web browsing by the user.

Microsoft Word Application Software Used to write and edit documents.

Windows 10/11 System Software Operating System — manages hardware and runs applications.

Antivirus (e.g., Windows System Software Protects the system from malware and Defender) runs in the background.

WhatsApp Desktop App Application Software Used for messaging and communication.

**11. application software. THEORY EXERCISE: Whatis the difference between system software and application software?**

**System Software**

**Definition:**

System software is a type of software designed to control and manage the computer hardware and create an environment for other software (like application software) to run.

**Purpose:**

Operates the system and provides a platform for running application software.

**Examples:**

1. Operating Systems (Windows, macOS, Linux)

2. Device Drivers

3. Utility Programs (Disk Management, Antivirus)

**Application Software**

**Definition:**

Application software is designed to help users perform specific tasks, like creating documents, browsing the web, or editing photos.

**Purpose:**

Directly serves the user’s needs for various personal or business tasks.

**Examples:**

1. Microsoft Word

2. Google Chrome

3. VLC Media Player

4. Adobe Photoshop

**Comparison Table**

**Feature System Software Application Software**

Function Manages system operations Performs specific user tasks

User Interaction Minimal (mostly in background) Directly used by the user

Installation Comes pre-installed (usually) Installed based on user need

Examples Windows OS, Antivirus, Drivers Word, Chrome, Zoom, Spotify

**Software Architecture**

**LAB EXERCISE:** Design a basic three-tier software architecture diagram for a web application.

**Three-Tier Architecture Diagram:**

+------------------------+

| Presentation Tier | ← Web browser / Mobile app

| (Frontend) |

| - HTML, CSS, JS |

| - React / Angular |

+------------------------+

│

▼

+------------------------+

| -Logic Tier | ← Server-side logic

| (Backend) |

| - PHP, Python, Node.js|

| - APIs, Controllers |

+------------------------+

│

▼

+------------------------+

| Data Tier | ← Database server

| (Database) |

| - MySQL, PostgreSQL |

| - MongoDB |

+------------------------+

**THEORY EXERCISE: Whatis the significance of modularity in software architecture?**

**Definition:**

Modularity in software architecture means breaking down an application into independent, self-contained components or modules, each responsible for a specific function.

**Significance / Benefits of Modularity:**

**1. Improved Maintainability**

1.Each module can be developed, tested, and updated independently.

2. Easier to fix bugs without affecting the entire system.

**2. Reusability**

1. Modules can be reused across different projects.

2. Example: An authentication module can be reused in multiple apps.

**3. Simplified Development**

1. Teams can work on different modules in parallel, speeding up development.

2. Easier to understand and manage code.

**4. Enhanced Security**

1. Sensitive functionalities (like login or payment) can be isolated in secure modules.

2. Reduces the risk of security issues spreading across the system.

**5. Scalability**

1. Individual modules can be scaled or upgraded independently as the application grows.

Modularity is crucial in software architecture because it makes applications easier to develop, maintain, test, reuse, and scale. It leads to cleaner, more organized, and more efficient systems.

**12.Layersin Software Architecture**

**LAB EXERCISE: Create a case study on the functionality of the presentation, business logic, and data access layers of a given software system.**

**Case Study:** Functionality of Presentation, Business Logic, and Data Access Layers

**1. Presentation Layer (UI Layer)**

**Function:**

This is the front-end of the application — it interacts with the user and displays content (buttons, forms, product listings, etc.).

**Technologies Used:**

HTML, CSS, JavaScript, React, Angular

**Example in Case Study:**

**1. The user opens the website and views a product catalog.**

They add items to the cart, click "Buy Now", or fill out a checkout form.

All actions here are collected and passed to the next layer.

**2. Business Logic Layer (Application Layer)**

**Function:**

Processes the rules and logic of the application. It acts as a middle layer, deciding how data is handled.

**Technologies Used:**

Java, Python, PHP, Node.js, .NET

**Example in Case Study:**

1. Checks if the product is in stock.

2. Calculates the total price, applies discounts, and verifies payment information.

3. Sends order confirmation or error messages to the UI.

**3. Data Access Layer (Persistence Layer)**

**Function:**

Manages all communication with the database — saving, retrieving, updating, or deleting data.

**Technologies Used:**

SQL, ORM tools (like Hibernate, Sequelize, Entity Framework), Database drivers

**Example in Case Study:**

1. Fetches product data when a user visits the catalog.

2. Stores order and payment details in the database.

3. Retrieves a customer’s past orders for their account page.

**How They Work Together:**

1. User clicks “Buy Now” →

2. UI sends request to Business Logic Layer →

3. Business Logic checks data and sends queries to Data Access Layer →

4. Data Access Layer interacts with the database →

5. Result flows back up to Presentation Layer and shows confirmation to user.

**Example System:**

1.Online Shopping Web Application (E-commerce Site)

**THEORY EXERCISE: Why are layers important in software architecture?**

**Definition:**

In software architecture, layers refer to separating the application into logical sections or levels (e.g., presentation, business logic, and data access) — each with a distinct responsibility.

**Importance of Layers:**

**1. Separation of Concerns**

1. Each layer focuses on a specific task (e.g., UI doesn’t handle database logic).

2. Makes the codebase easier to understand and manage.

**2. Maintainability**

1. Changes in one layer (e.g., updating the UI) can be done without affecting others.

2. Easier to fix bugs or add features.

**3. Reusability**

Layers like business logic can be reused across multiple applications (e.g., web, mobile).

**4. Testability**

Individual layers can be unit tested independently.

For example, you can test business rules without needing a user interface.

**5. Scalability**

Each layer can be scaled separately. For instance, the database layer can be optimized or moved to a separate server.

**6. Security**

Sensitive operations (like authentication or data access) are separated from the UI, reducing the risk of exposing vulnerabilities.

Layers in software architecture promote clean, organized, and scalable design. They help developers build robust, maintainable, and testable applications by dividing responsibilities across well-defined sections.

**13. Software Environments**

**LAB EXERCISE:**

Explore different types of software environments (development, testing, production). Set up a basic environment in a virtual machine.

Types of Environments:

- Development Environment: Where developers write and test their code locally.

- Testing Environment: Used for quality assurance, bug testing, and validation.

- Production Environment: The live system used by end users.

Setup Example:

I used VirtualBox to install Ubuntu Linux. Installed Python and Git to create a simple development environment. Configured a sample Flask web application to demonstrate how different environments can be managed.

**THEORY EXERCISE:**

The development environment is important in software production because it provides a safe and controlled space for programmers to write, debug, and test code before moving it to testing or production. It reduces risks, ensures stability, and helps maintain productivity.

**14. Source Code**

**LAB EXERCISE:**

Write and upload your first source code file to GitHub.

Steps:

1. Create a simple C program (Hello World).

2. Initialize a local Git repository.

3. Commit the file and push it to GitHub.

Example Code (hello.c):

#include<stdio.h>

int main() {

printf("Hello, World!\n");

return 0;

}

**THEORY EXERCISE:**

Source code is the human-readable set of instructions written in programming languages such as C, Python, or Java. Machine code, on the other hand, is the binary code (0s and 1s) that the computer's processor understands. Source code must be compiled or interpreted into machine code before execution.

**15. GitHub and Introductions**

**LAB EXERCISE:**

Create a GitHub repository and document how to commit and push code changes.

Steps:

1. Create a repository on GitHub.

2. Clone the repository using: git clone <repo\_url>

3. Add files: git add filename

4. Commit changes: git commit -m "Initial commit"

5. Push changes: git push origin main

**THEORY EXERCISE:**

Version control is important because it allows multiple developers to work collaboratively, tracks changes to code, prevents overwriting, and maintains a history of project evolution. It also enables reverting to previous versions when errors occur.

**16. Student Account in GitHub**

**LAB EXERCISE:**

Create a student account on GitHub and collaborate on a small project with a classmate.

Steps:

1. Sign up with GitHub Student Developer Pack.

2. Create a repository and add a collaborator.

3. Both students clone the repo and work on different features.

4. Merge changes using pull requests.

**THEORY EXERCISE:**

GitHub provides students with free access to tools, collaboration features, and resources. It helps students learn real-world development workflows, showcase projects, and collaborate easily with peers.

**17. Types of Software**

**LAB EXERCISE:**

List of regularly used software and classification:

System Software:

- Windows 11, Linux Ubuntu

Application Software:

- MS Word, Google Chrome, WhatsApp

Utility Software:

- Antivirus (Avast), WinRAR, Disk Cleanup

**THEORY EXERCISE:**

Open-source software is software whose source code is available publicly for free use, modification, and distribution. Examples: Linux, Apache.

Proprietary software is owned by companies and requires a license to use, with restrictions on modification. Examples: Microsoft Office, Adobe Photoshop.

**18. Git and GitHub Training**

**LAB EXERCISE:**

Followed a Git tutorial:

- Cloned a repository using git clone.

- Created a new branch using git checkout -b feature.

- Merged the branch into main using git merge feature.

**THEORY EXERCISE:**

Git improves collaboration by allowing multiple developers to work on the same project simultaneously, track changes, manage branches for new features, and merge work seamlessly. It prevents code conflicts and maintains history.

**19. Application Software**

**LAB EXERCISE:**

Report on Types of Application Software and Productivity:

- Word Processing Software (MS Word, Google Docs): Helps create documents quickly.

- Spreadsheet Software (MS Excel, Google Sheets): Used for data analysis and financial management.

- Presentation Software (PowerPoint, Canva): Improves communication of ideas.

- Database Software (MySQL, MS Access): Helps organize and retrieve information.

- Communication Software (Zoom, Slack): Facilitates remote collaboration.

These applications improve productivity by automating tasks, increasing efficiency, and enabling teamwork.

**THEORY EXERCISE:**

Application software plays a critical role in businesses by enabling employees to complete tasks efficiently, manage resources, analyze data, and communicate effectively. It supports decision-making, improves workflow, and enhances customer service.

**20. Software Development Process**

**LAB EXERCISE:**

Flowchart of the Software Development Life Cycle (SDLC):

1. Requirement Analysis → 2. System Design → 3. Implementation → 4. Testing → 5. Deployment → 6. Maintenance

The flowchart represents the step-by-step process followed during software development.

THEORY EXERCISE:

The main stages of the software development process are:

1. Requirement Analysis

2. System Design

3. Implementation (Coding)

4. Testing

5. Deployment

6. Maintenance

These stages ensure structured and efficient software creation.

**21. Software Requirement**

**LAB EXERCISE:**

Requirement Specification for a Library Management System:

- The system should allow users to search for books by title, author, or ISBN.

- Admins can add, update, or remove books.

- Users can borrow and return books.

- The system should track due dates and calculate late fees.

- Generate reports on book availability and borrowing history.

**THEORY EXERCISE:**

Requirement analysis is critical because it ensures developers fully understand what the client needs. It reduces misunderstandings, prevents costly rework, and provides a clear foundation for design and development.

**22. Software Analysis**

**LAB EXERCISE:**

Functional Analysis for an Online Shopping System:

- User registration and login

- Browse products by categories

- Add items to shopping cart

- Checkout and payment processing

- Order history tracking

- Admin panel for product management and order monitoring

**THEORY EXERCISE:**

Software analysis identifies functional and non-functional requirements of a system. It helps in defining what the system should do, improving design decisions, and ensuring the final product meets user expectations.

**23. System Design**

**LAB EXERCISE:**

Basic System Architecture for a Food Delivery App:

- Client Side (Mobile App / Website): User registration, browsing restaurants, placing orders.

- Server Side: Processes orders, manages restaurants, tracks delivery.

- Database: Stores user details, restaurant menus, order history.

- Delivery Module: Assigns delivery agents and provides real-time tracking.

**THEORY EXERCISE:**

Key elements of system design include:

- Architecture design

- Database design

- User interface design

- Security considerations

- Integration and scalability planning

**24. Software Testing**

**LAB EXERCISE:**

Test Cases for a Simple Calculator Program:

1. Input: 2 + 3 → Expected Output: 5

2. Input: 10 - 4 → Expected Output: 6

3. Input: 5 \* 6 → Expected Output: 30

4. Input: 8 / 2 → Expected Output: 4

5. Input: 9 / 0 → Expected Output: Error (division by zero)

**THEORY EXERCISE:**

Software testing is important because it ensures the software is free from critical bugs, meets requirements, improves quality, and prevents failures in production. It also increases customer trust and reduces costs of fixing errors later.

**25. Maintenance**

**LAB EXERCISE:**

Real-world Case: WhatsApp introduced multi-device login, which required critical maintenance. The feature was added through updates, bug fixes, and performance improvements to ensure smooth operation.

**THEORY EXERCISE:**

Types of Software Maintenance:

- Corrective: Fixing bugs and errors.

- Adaptive: Updating the system to work with new hardware or OS.

- Perfective: Improving performance or adding new features.

- Preventive: Enhancing reliability to prevent future issues.

**26. Development**

**THEORY EXERCISE:**

Differences between Web and Desktop Applications:

- Web Applications:

\* Run in a browser.

\* Accessible from anywhere with internet.

\* Easier to update centrally.

\* Examples: Gmail, Google Docs.

- Desktop Applications:

\* Installed on a personal computer.

\* Work offline without internet.

\* Updates must be installed by users.

\* Examples: MS Word, Photoshop.

**27. Web Application**

**THEORY EXERCISE:**

What are the advantages of using web applications over desktop applications?

Answer:

• Accessibility: Web apps can be accessed from anywhere with an internet connection.

• Cross-Platform Compatibility: They run on multiple operating systems and devices via browsers.

• Easy Updates: Updates are applied on the server side, so users automatically get the latest version.

• Cost-Effective: No installation is required on individual machines.

• Collaboration: Web apps enable real-time sharing and teamwork.

• Cloud Storage: Data is stored online, reducing dependency on local hardware.

**28. Designing**

**THEORY EXERCISE:**

What role does UI/UX design play in application development?

Answer:

• UI (User Interface): Deals with the design, layout, and look of the application.

• UX (User Experience): Focuses on usability, navigation, and how users interact with the system.

Role in Development:

• Improves ease of use and accessibility.

• Enhances customer satisfaction and engagement.

• Reduces user errors with intuitive design.

• Supports branding and creates a professional impression.

**29. Mobile Application**

**THEORY EXERCISE: What are the differences between native and hybrid mobile apps?**

Feature

Native Apps

Hybrid Apps

Platform

Built for a single platform (Android or iOS).

Runs on multiple platforms with a single codebase.

Languages Used

Java/Kotlin (Android), Swift/Objective-C (iOS).

HTML, CSS, JavaScript with frameworks like Ionic or React Native.

Performance

High (optimized for the platform).

Moderate (depends on web view wrapper).

Access to Features

Full access to device hardware and OS features.

Limited access via plugins or APIs.

Development Cost

Higher (separate code for each platform).

Lower (single codebase for multiple platforms).

User Experience

Superior, smooth, and responsive.

Slightly less fluid compared to native.

Update Process

Requires app store approval and separate updates.

Easier updates, often through server-side changes.

**31. Desktop Application**

**LAB EXERCISE: Build a Simple Desktop Calculator Application Using a GUI Library**

Example in Python using Tkinter:

import tkinter as tk

def click(event):

text = event.widget.cget("text")

if text == "=":

try:

result = eval(str(entry.get()))

entry.delete(0, tk.END)

entry.insert(tk.END, result)

except Exception as e:

entry.delete(0, tk.END)

entry.insert(tk.END, "Error")

elif text == "C":

entry.delete(0, tk.END)

else:

entry.insert(tk.END, text)

root = tk.Tk()

root.title("Calculator")

entry = tk.Entry(root, font="Arial 20")

entry.pack(fill=tk.BOTH, ipadx=8)

buttons = ["7", "8", "9", "/", "4", "5", "6", "\*",

"1", "2", "3", "-", "C", "0", "=", "+"]

f = tk.Frame(root)

for i, b in enumerate(buttons):

btn = tk.Button(f, text=b, font="Arial 15", height=2, width=5)

btn.grid(row=i//4, column=i%4)

btn.bind("<Button-1>", click)

f.pack()

root.mainloop()

📘 THEORY EXERCISE: Pros and Cons of Desktop Applications Compared to Web Applications

Aspect Desktop Applications Web Applications

Installation Must be installed on each machine No installation; accessed via browser

Offline Access Can work offline Requires internet connection

Performance Usually faster due to local resources Depends on network speed

Maintenance Harder to update (manual updates per user) Easy to update (centralized)

Compatibility OS-specific (Windows, macOS) Cross-platform (works on any browser)

32. Flowchart

🧪 LAB EXERCISE: Draw a Flowchart Representing the Logic of a Basic Online Registration System

📝 Steps to Represent in Flowchart:

Start

Input user details (name, email, password)

Validate input (e.g., email format, password strength)

If validation fails → Show error → Go back to input

If validation passes → Save data to database

Show success message

End

📌 Use standard flowchart symbols:

Ovals for Start/End

Parallelogram for Input/Output

Rectangles for Processes

Diamonds for Decision-making